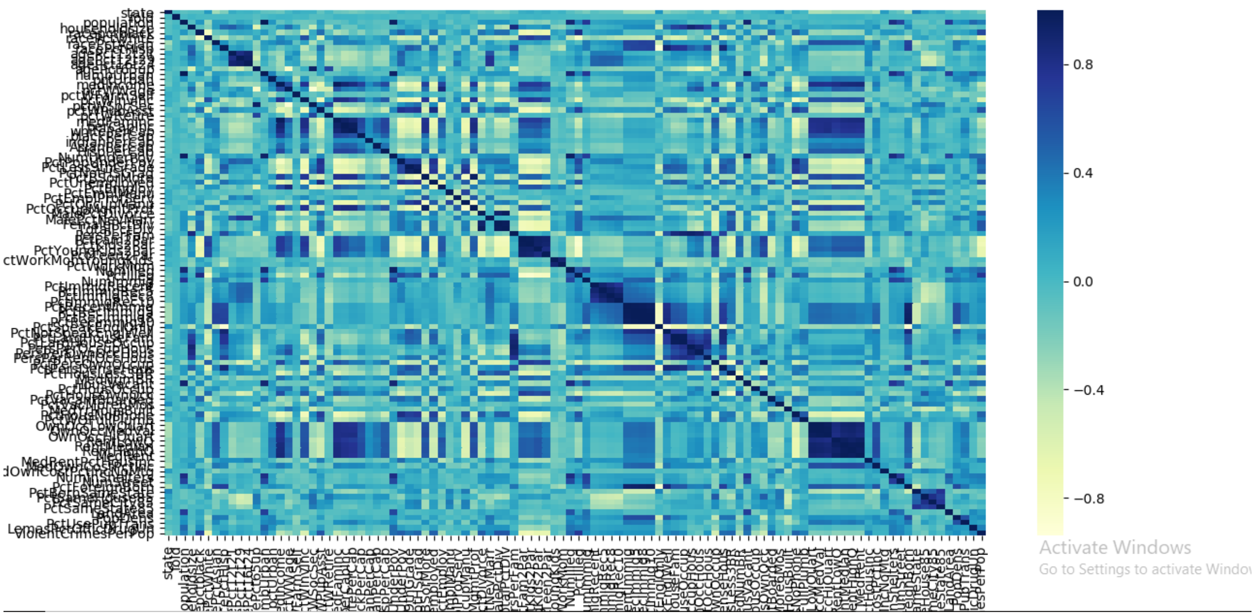
Question 1

b. Data Imputation

I have utilized .replace to edit the missing values as zero.

(c) Correlation Matrix



(d) Coeff- CV

I calculated the sample standard deviation and sample mean for each feature using dataframe.std() and datafram.mean() respectively.

I then used the formula ***CV=s/m***

I received the following results:

CV values features:

state 0.571671

fold 0.523062

population	2.203503
householdsize	0.353298
racepctblack	1.410920
racePctWhite	0.323782
racePctAsian	1.359162
racePctHisp	1.614278
agePct12t21	0.365840
agePct12t29	0.290693
agePct16t24	0.495161
agePct65up	0.423442
numbUrban	2.001744
pctUrban	0.638849
medIncome	0.579753
pctWWage	0.327710
pctWFarmSelf	0.700030
pctWInvInc	0.359240
pctWSocSec	0.368513
pctWPubAsst	0.699031
pctWRetire	0.349639
medFamInc	0.527732
perCapInc	0.545633
whitePerCap	0.507552
blackPerCap	0.589469
indianPerCap	0.809685
AsianPerCap	0.606194
HispPerCap	0.473960
NumUnderPov	2.304970
PctPopUnderPov	0.753980
	...

HousVacant	1.958780
PctHousOccup	0.269647
PctHousOwnOcc	0.337541
PctVacantBoarded	1.064742
PctVacMore6Mos	0.436119
MedYrHousBuilt	0.470411
PctHousNoPhone	0.918211
PctWOFullPlumb	0.848744
OwnOccLowQuart	0.847880
OwnOccMedVal	0.878750
OwnOccHiQuart	0.874733
RentLowQ	0.633186
RentMedian	0.561884
RentHighQ	0.587014
MedRent	0.555592
MedRentPctHousInc	0.345830
MedOwnCostPctInc	0.416391
MedOwnCostPctIncNoMtg	0.476933
NumInShelters	3.485481
NumStreet	4.407702
PctForeignBorn	1.072291
PctBornSameState	0.335575
PctSameHouse85	0.338944
PctSameCity85	0.320105
PctSameState85	0.304240
LandArea	1.678031
PopDens	0.872187
PctUsePubTrans	1.416673
LemasPctOfficDrugUn	2.555266

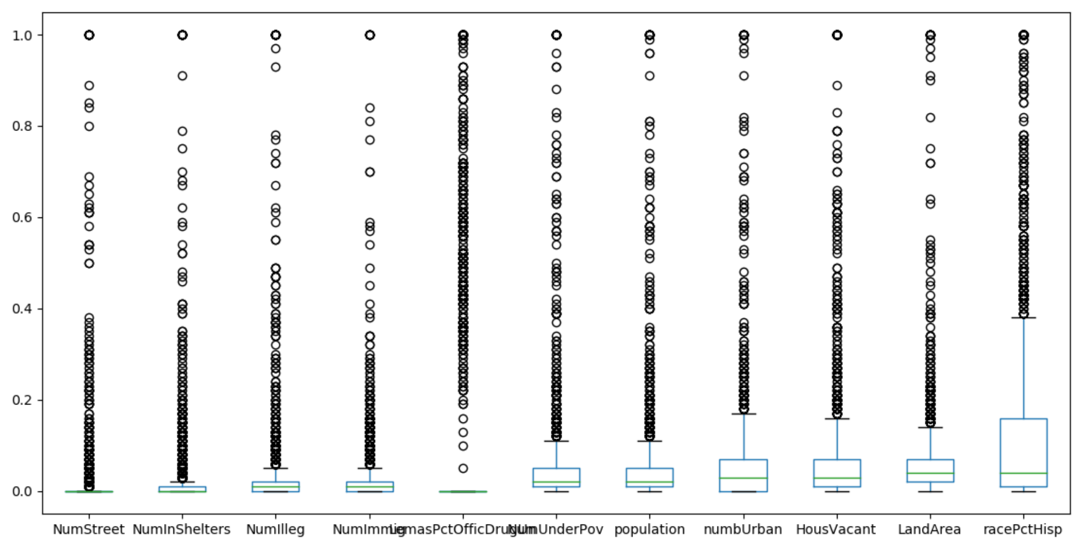
ViolentCrimesPerPop 0.979015

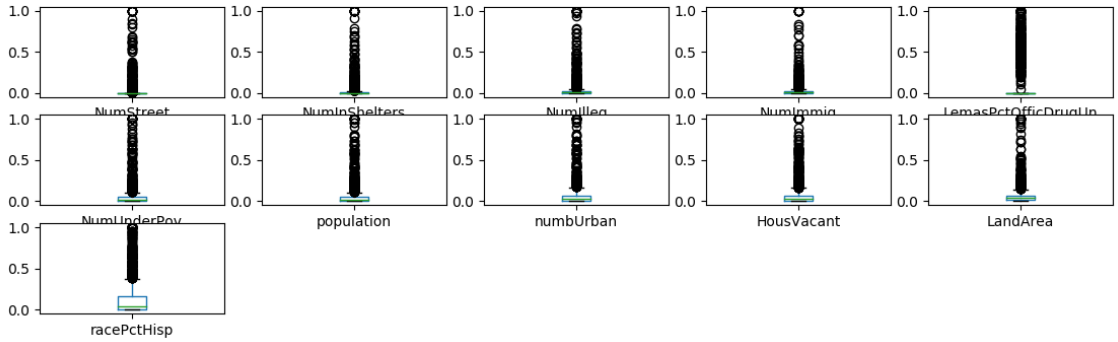
(e) Best $\lfloor\sqrt{128}\rfloor$ features with highest CV

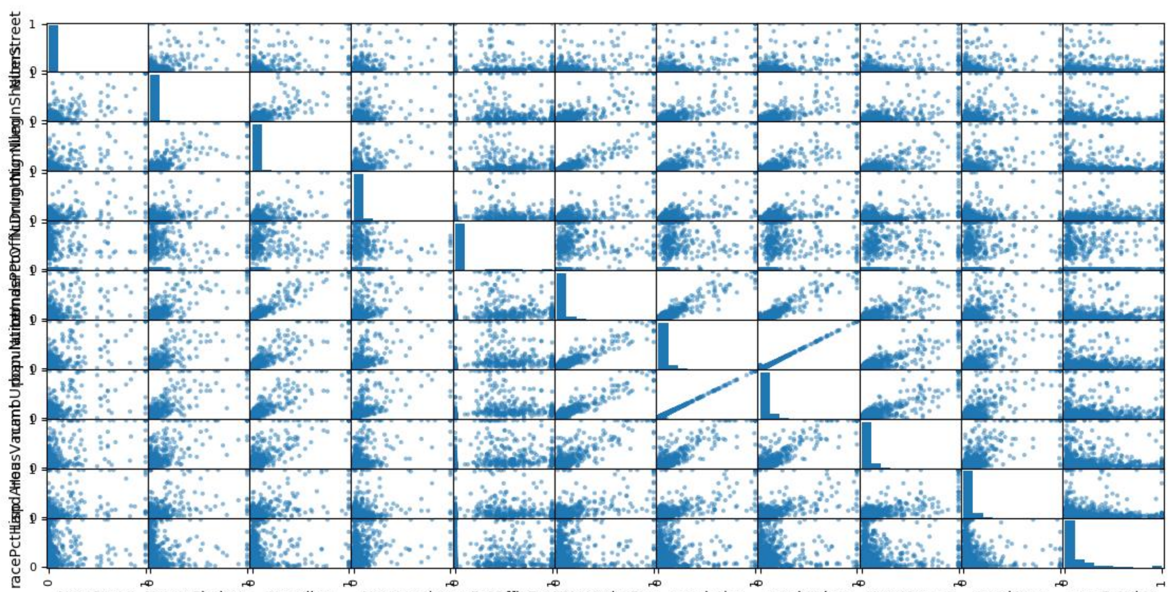
Top 11 features:

['NumStreet', 'NumInShelters', 'NumIlleg', 'NumImmig', 'LemasPctOfficDrugUn', 'NumUnderPov', 'population', 'numbUrban', 'HousVacant', 'LandArea', 'racePctHisp']

I plotted the box plots and the scatterplot for this and received the following results:







From the scatter plot I can conclude that numbUrban and population are correlated.

(f) Linear Model

The Test error observed is:

Test Error: 0.017325523786361815

(g) Ridge Regression

I used RidgeCV and a linspace array of 100 elements to try as lambda

```
arr=np.linspace(0.01,100.1,100)
```

The following results are obtained:

Score: 0.6450746251885445

Test Error: 0.01687858288302621

penalty 3.0430303030303025

So the best **lambda value is 3.043030.**

(h) Lasso CV without Normalization

I used LassoCV and again used a linspace to determine the best try-out value of lambda.

If the regression Co-ef is not zero means that feature is selected.

I used this piece of code to deduce that and print them.

```
final_features=model.coef_

if(final_features[i]!=0)
```

I obtained the following results:

Score: 0.6419834599473726

Test Error: 0.017025584175216057

penalty 0.001

state : -0.0010259595847256992

county : -0.00011054721167369276

community : -3.1541730032138775e-07

fold : -0.0023348142097352432

racepctblack : 0.2073661665744953

pctUrban : 0.03655184382092607

pctWPubAsst : 0.028607156154093198

AsianPerCap : 0.0033265146987917953

MalePctDivorce : 0.10643847237352005

PctKids2Par : -0.20581716995591107

PctYoungKids2Par : -0.015740606752911706

PctWorkMom : -0.05002839749472875

PctIlleg : 0.16739626773490537

PctRecImmig10 : 0.019786793040330933

PctPersDenseHous : 0.13602842432850787

PctHousLess3BR : 0.005114859756699327

HousVacant : 0.08507829673038414

PctHousOccup : -0.04396040604653588

PctVacantBoarded : 0.05161371067067361

NumStreet : 0.07038053081560751

PctForeignBorn : 0.005120179734442512

PctSameCity85 : 0.0052217197159349655

PolicReqPerOffic : 0.005606927729461414

PopDens : 0.003248710430474636

LemasPctPolicOnPatr : 0.015207088652245264

LemasGangUnitDeploy : 0.019762942866336456

Features Shortlisted: 26

So lassoCV determines lambda=0.001 as the best value and shortlists 26 features.

LASSO with Normalization

I toggled the *normalize* parameter in LassoCV as True and obtained the following results:

Score: 0.5841993191395096

Test Error: 0.01977352636015587

penalty 0.001

racePctWhite : -0.16657517842272596

PctKids2Par : -0.34680770443061393

PctIlleg : 0.16458957908224858

HousVacant : 0.06341866794866101

Number of important features: 4

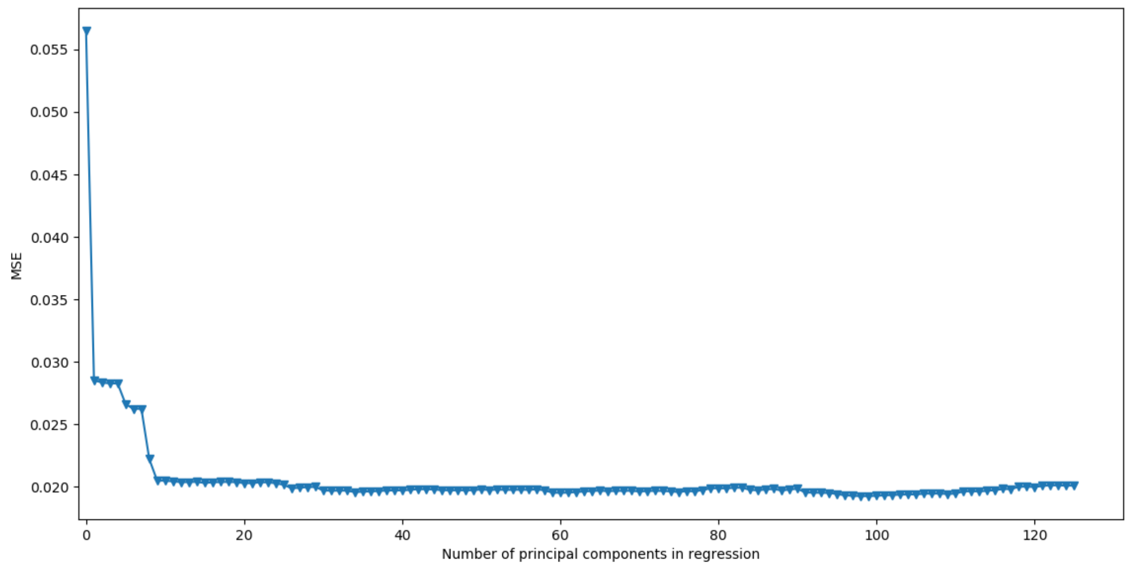
The change is clearly evident the feature list comes down to just 4 but the test error increases ultimately decreasing the score.

(i) PCR with CV

I ran PCA to determine the best value of M and used 5-fold CV in the process.
I then plugged them into a dictionary and sorted them based on the MSE to enable me to pick the M value that had the least error.

I then used this value of M in a linear model to transform the test set and perform the regression task.

I obtained the following results:



Dictionary of M values:

dictionary OrderedDict([(99, 0.019255872002397124), (98, 0.019256843223897348), (97, 0.0192810388514019), (100, 0.01928143126711105), (101, 0.01931658815612832), (96, 0.019319580011890845), (102, 0.019348271866247393), (103, 0.019370916389839277), (105, 0.019389379498006162), (104, 0.01939068094609972), (95, 0.0194217597254687 3), (109, 0.019422217374775674), (107, 0.01945454959657155), (106, 0.019455784434287547), (110, 0.019460674809187523), (108, 0.019463803122342964), (94, 0.01950996072731243), (60, 0.01951845243345194), (93, 0.019522077658924476), (59, 0.019532442719181014), (92, 0.019532930782969766), (61, 0.019536363410078705), (91, 0.01954245387129915), (34, 0.019576413753595236), (62, 0.019582340974232244), (75, 0.01958236494238703), (35, 0.019611699096278593), (63, 0.019626782467375307), (70, 0.019629760863401218), (36, 0.019631916713676013), (77, 0.019632162116857817), (74, 0.019633403401009 36), (113, 0.0196367944104123), (111, 0.019637203393157 78), (112, 0.019645994263317 54), (66, 0.019648053653579335), (71, 0.019659409601898475), (76, 0.019663241664918736), (37, 0.019663296213774224), (64, 0.019667274909831188), (73, 0.019669121158559757), (68, 0.019674401782192694), (67, 0.019677354767598635), (114, 0.019679845534100113), (72, 0.01968113890225487), (31, 0.019683672957158005), (69, 0.01968580352394383), (33, 0.019695195549115353), (32, 0.019695571329062973), (65, 0.019697873679106394), (38, 0.01970010723472491), (88, 0.019701635979315506), (39, 0.019712315254459115), (30, 0.019714604310302135), (40, 0.019717516993763118), (46, 0.019718264394675618), (85, 0.0197217145062874), (48, 0.019721715004876107), (58, 0.019722191703782112), (47, 0.019723627752405583), (49, 0.019728070677122664), (45, 0.019731258286651183), (115, 0.019731438014422113), (51, 0.019743174325460827), (78, 0.019743314043476255), (41, 0.019755134660591745), (84, 0.019755164557210752), (56, 0.019757048331832887), (54, 0.019757372258845808), (53, 0.019757590524552216), (55, 0.019770145744787374), (52, 0.019776299458438857), (89, 0.01978665181609933), (86, 0.0197903854619682), (50, 0.019790704915580482), (44, 0.019793404059860603), (57, 0.019797233872859095), (43, 0.01980627739020558), (42, 0.019812244499313243), (117, 0.01981549215493037), (116, 0.019830720528872227), (87, 0.019831200495918886), (90, 0.019848007683908892), (81, 0.019873639452373464), (79, 0.019877250453991347), (26, 0.019885143394946295), (80, 0.01988794177614182), (27, 0.019922539453053993), (82, 0.0199236631319499), (83, 0.019929961201371194), (28, 0.019981019901742548), (120, 0.01998458677459205), (29, 0.019990018781394094), (118, 0.020007972794674173), (119, 0.020014288968558242), (121, 0.02007527804545555), (124, 0.02009015669690114), (125, 0.02010055568454734), (122, 0.02011702177517309), (123, 0.02011801449353835), (25, 0.020163894327046412), (24, 0.020282996708610026), (20, 0.020293570495375356), (21, 0.02029812873234818), (15, 0.02030493547346702), (16, 0.02035914049404846), (22, 0.02035973925042847), (23, 0.02036253614903158), (19, 0.020370868325285514), (12, 0.020374659620921286), (13, 0.02038085096679267), (17, 0.020386482578913463), (18, 0.020395264969361154), (14, 0.020403690416886913), (11, 0.020434267840832153), (9, 0.020485634468585413), (10, 0.020496410072339814), (8, 0.022211087412476845), (6, 0.026235608385106184), (7, 0.026249398494549957), (5, 0.02662644615660607), (4, 0.028273241179515417), (3, 0.028278349083876542), (2, 0.028346828684540765), (1, 0.02853284043159319)])

M: 99

Into Linear Model Fitting
Error: 0.01803019592104647
Score: 0.6208583333357408

(j) **XGBoost with L1-penalized gradient boosting**

I used an XGBRegressor and GridSearchCV with 5-fold CV to complete this problem.

I again passed an array of alphas for the GridSearchCV to try out and output.

I obtained the following results:

In [4]:
```
import xgboost as xgb
from sklearn.grid_search import GridSearchCV
import pandas as pd
import numpy as np

df_train=pd.read_csv("arpit_communities.csv")#skiprows=20,index_col=21)
df_train.replace('na',0,inplace=True)
df_train.replace('?',0,inplace=True)

X_train=df_train.values[:,1:171]
Y_train=df_train.values[:,:1]

optimized_GBM = GridSearchCV(cv=5,
       estimator=xgb.XGBRegressor(),
       param_grid={'reg_alpha': np.linspace(np.float_power(10, -4), np.float_power(10, 1), 20)},
        refit=True, scoring='neg_mean_squared_error', verbose=1)
# Optimize for accuracy since that is the metric used in the Adult Data Set notation
optimized_GBM.fit(X_train, Y_train)

print(optimized_GBM.grid_scores_)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:   38.1s finished
[mean: -130.71711, std: 9.22320, params: {'reg_alpha': 0.0001}, mean: -130.92072, std: 9.01665, params: {'reg_alph
a': 0.5264105263157894}, mean: -130.58151, std: 8.71732, params: {'reg_alpha': 1.052721052631579}, mean: -132.1493
5, std: 9.08927, params: {'reg_alpha': 1.5790315789473683}, mean: -130.38975, std: 8.67986, params: {'reg_alpha':
2.105342105263158}, mean: -131.17741, std: 9.62229, params: {'reg_alpha': 2.6316526315789477}, mean: -130.81555, st
d: 10.14610, params: {'reg_alpha': 3.157963157894737}, mean: -130.68021, std: 8.12625, params: {'reg_alpha': 3.6842
73684210526}, mean: -131.68327, std: 9.82658, params: {'reg_alpha': 4.210584210526315}, mean: -130.78487, std: 9.34
996, params: {'reg_alpha': 4.736894736842105}, mean: -129.66058, std: 8.52199, params: {'reg_alpha': 5.263205263157
895}, mean: -129.30700, std: 10.05831, params: {'reg_alpha': 5.7895157894736835}, mean: -129.73098, std: 9.23980, p
arams: {'reg_alpha': 6.315826315789473}, mean: -130.17766, std: 7.11517, params: {'reg_alpha': 6.842136842105263},
mean: -130.34875, std: 10.57276, params: {'reg_alpha': 7.368447368421052}, mean: -128.85229, std: 7.86663, params:
{'reg_alpha': 7.894757894736841}, mean: -129.96006, std: 9.14826, params: {'reg_alpha': 8.421068421052631}, mean: -
129.95042, std: 8.98339, params: {'reg_alpha': 8.947378947368842}, mean: -130.04527, std: 9.92925, params: {'reg_alp
ha': 9.473689473684421}, mean: -129.40500, std: 9.33109, params: {'reg_alpha': 10.0}]
```

In [6]:
```
print(optimized_GBM.best_params_)
```

```
{'reg_alpha': 7.894757894736841}
```

The best value of alpha was: 7.89476

Question 2

(b) (i)

Imputation can be done using several methods:

Fillna, backfillna, Imputer package in sklearn, or by simply just a replace.

I have used replace to accomplish this and convert the entire data into string using astype().

I also changed the Class Neg: 0 and Class Pos:1

(ii) **CV**

This was done in a similar fashion as to the question 1.

The following results were obtained:

[[244.88836426184145 'cf_000']
[244.51076535063208 'co_000']
[244.37598592582142 'ad_000']
[237.93055371566217 'cs_009']
[123.21609721755667 'dh_000']
[117.49422514513171 'dj_000']
[92.91775503608642 'ag_000']
[87.33249956499247 'as_000']
[84.73373459937712 'ay_009']
[80.42497540906561 'ak_000']
[77.83854428850402 'az_009']
[77.4538571344374 'ch_000']
[68.88275094860896 'au_000']
[58.07807152884354 'cr_000']
[52.82471400357465 'ay_001']
[52.292813588128965 'df_000']
[51.3322277688843 'dz_000']
[49.36665925628379 'ef_000']
[48.220079107190436 'cs_008']
[44.26599598905513 'aj_000']
[42.48174746009007 'eg_000']
[39.73908782176686 'dl_000']
[39.24865364892385 'ay_002']
[38.48616191142954 'dg_000']
[37.42828471068997 'ay_000']

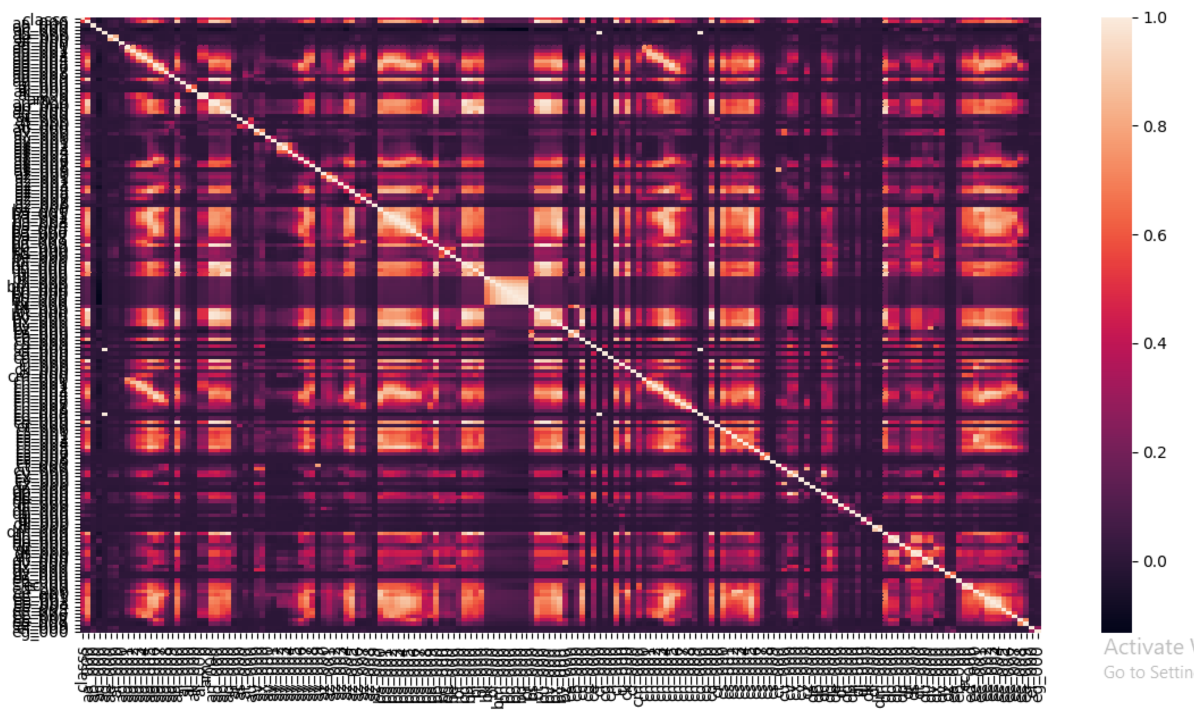
[37.03912307273606 'dk_000']
[36.60090813287275 'cy_000']
[36.26140326564316 'dm_000']
[35.24931353569572 'ag_001']
[34.94651895491634 'ea_000']
[34.27327078908045 'cn_009']
[33.75234540476702 'ay_004']
[33.35756746889936 'ag_009']
[29.367526519061453 'da_000']
[28.735090223602636 'ay_003']
[26.335574345711866 'cn_000']
[24.200136597481663 'ae_000']
[23.708186710574548 'at_000']
[22.679650237535288 'az_008']
[22.030103070594105 'dq_000']
[19.4712950562839 'af_000']
[18.203806264011888 'ai_000']
[17.56590713284474 'ag_002']
[16.229426743966997 'az_007']
[14.970729682408079 'cl_000']
[14.55100707851276 'cz_000']
[13.949635339265187 'cp_000']
[13.290748537881425 'az_002']
[12.524654611577459 'ay_005']
[11.826232152583813 'di_000']
[11.35434651281637 'ar_000']
[11.26256085031939 'cn_001']
[11.069531465621136 'cj_000']
[10.383493866617679 'ab_000']
[9.744570095082587 'cn_008']
[9.434445752852419 'az_000']
[9.430241407037293 'ba_009']
[9.173106315332872 'al_000']
[9.155221404503994 'am_0']
[8.880859131119804 'az_006']
[8.647402475930393 'ag_003']
[8.516292469155456 'ct_000']
[7.796648367537321 'dy_000']
[7.733630366859148 'az_001']
[7.681209758219827 'classs']
[7.530939390242064 'az_003']
[7.462350803533416 'bf_000']
[7.274336193370247 'bc_000']
[7.134734818543998 'cu_000']

[6.887037454948392 'be_000']
[6.8654458053286405 'dr_000']
[6.830710218280034 'ba_008']
[6.70740290737369 'cn_002']
[6.346280207398323 'cn_007']
[6.316860672188324 'bz_000']
[6.259495274867568 'db_000']
[6.225098758984985 'ag_008']
[6.033640551734054 'av_000']
[5.691612313066102 'ee_009']
[5.463603548437661 'ag_004']
[5.450834018431791 'cs_007']
[5.4180729882698895 'cm_000']
[5.374132379126713 'dx_000']
[5.371019480888662 'bd_000']
[5.119683093431874 'cs_002']
[5.019734094778544 'ee_007']
[4.935251999594238 'cx_000']
[4.717482545409535 'cg_000']
[4.696075538143602 'cs_004']
[4.613376473854331 'de_000']
[4.561663794029881 'eb_000']
[4.195532267195919 'cn_003']
[4.051600091961907 'ax_000']
[3.8198300949707136 'ay_008']
[3.68299249714019 'cs_001']
[3.5968313156621043 'dv_000']
[3.5872346650103997 'bj_000']
[3.323088393431945 'ay_007']
[3.314736519414211 'ee_000']
[3.298913993651963 'ee_001']
[3.260185601566189 'ee_008']
[3.22998123654064 'ee_006']
[3.1722971106568476 'cn_006']
[3.1455577252377824 'cs_003']
[3.1189351223710817 'dd_000']
[3.0939997750845376 'ap_000']
[3.0623165631663043 'ck_000']
[3.059127327981951 'ay_006']
[3.0440620186488476 'ba_006']
[3.043953611397488 'az_005']
[3.0339490484047236 'bi_000']
[3.0269475459043695 'br_000']
[2.971601774161997 'bq_000']

[2.9621066820642303 'ag_005']
[2.925292934460321 'du_000']
[2.9141392916849345 'ba_002']
[2.9056200333435585 'ec_00']
[2.903214785061223 'dn_000']
[2.8705151562076527 'bp_000']
[2.869491603335408 'aq_000']
[2.867503049354326 'ag_007']
[2.8637008957828605 'ee_005']
[2.8509856819000663 'az_004']
[2.8450073344514903 'ba_007']
[2.749732187640745 'ba_003']
[2.7411691910958322 'bo_000']
[2.7163353472921985 'ba_000']
[2.713100046440876 'ba_005']
[2.6828506487544375 'ed_000']
[2.6485568572349028 'ba_004']
[2.6430907143606017 'bh_000']
[2.6411890158096427 'ba_001']
[2.6380240917274773 'ee_004']
[2.63436991391018 'cn_004']
[2.6133481412656425 'cc_000']
[2.6106577147106385 'ee_002']
[2.5995630658141926 'bx_000']
[2.590124055687687 'ee_003']
[2.552650132444224 'bn_000']
[2.527568168504519 'cs_005']
[2.462328281127565 'by_000']
[2.451896061495529 'bt_000']
[2.450937577943998 'aa_000']
[2.421498334642216 'bu_000']
[2.4214981698459077 'bv_000']
[2.4214981359196712 'cq_000']
[2.420562031330786 'bb_000']
[2.4082840520729047 'ci_000']
[2.386890809599732 'ds_000']
[2.3738311833499317 'ag_006']
[2.3609029920656384 'cn_005']
[2.327518597655898 'ah_000']
[2.3250156589461577 'bg_000']
[2.310240723932622 'ac_000']
[2.2847268999382497 'ao_000']
[2.2774271782222115 'ce_000']
[2.2653994478781003 'an_000']

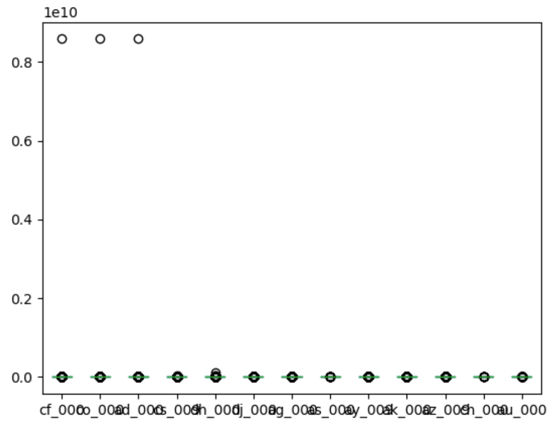
[2.2566021209926292 'dt_000']
[2.2312129246168433 'bm_000']
[2.2311750209151135 'cv_000']
[2.2099856627107117 'do_000']
[2.197618552624876 'dc_000']
[2.1550953733897757 'cs_006']
[2.0642138575664157 'dp_000']
[1.8957196403512118 'cs_000']
[1.625658055074731 'bl_000']
[1.4256062927273956 'bk_000']
[1.0638627687031057 'bs_000']
[1.0136149669591161 'ca_000']
[0.9228512609832727 'cb_000']
[0.10674849332694764 'cd_000']]

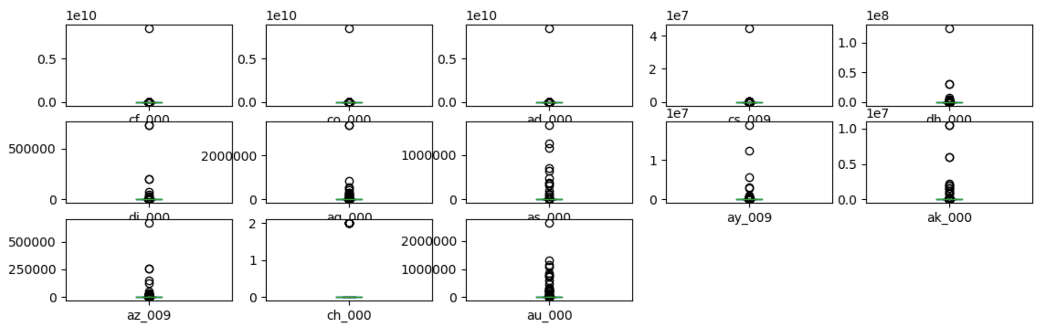
(iii) Correlation Matrix



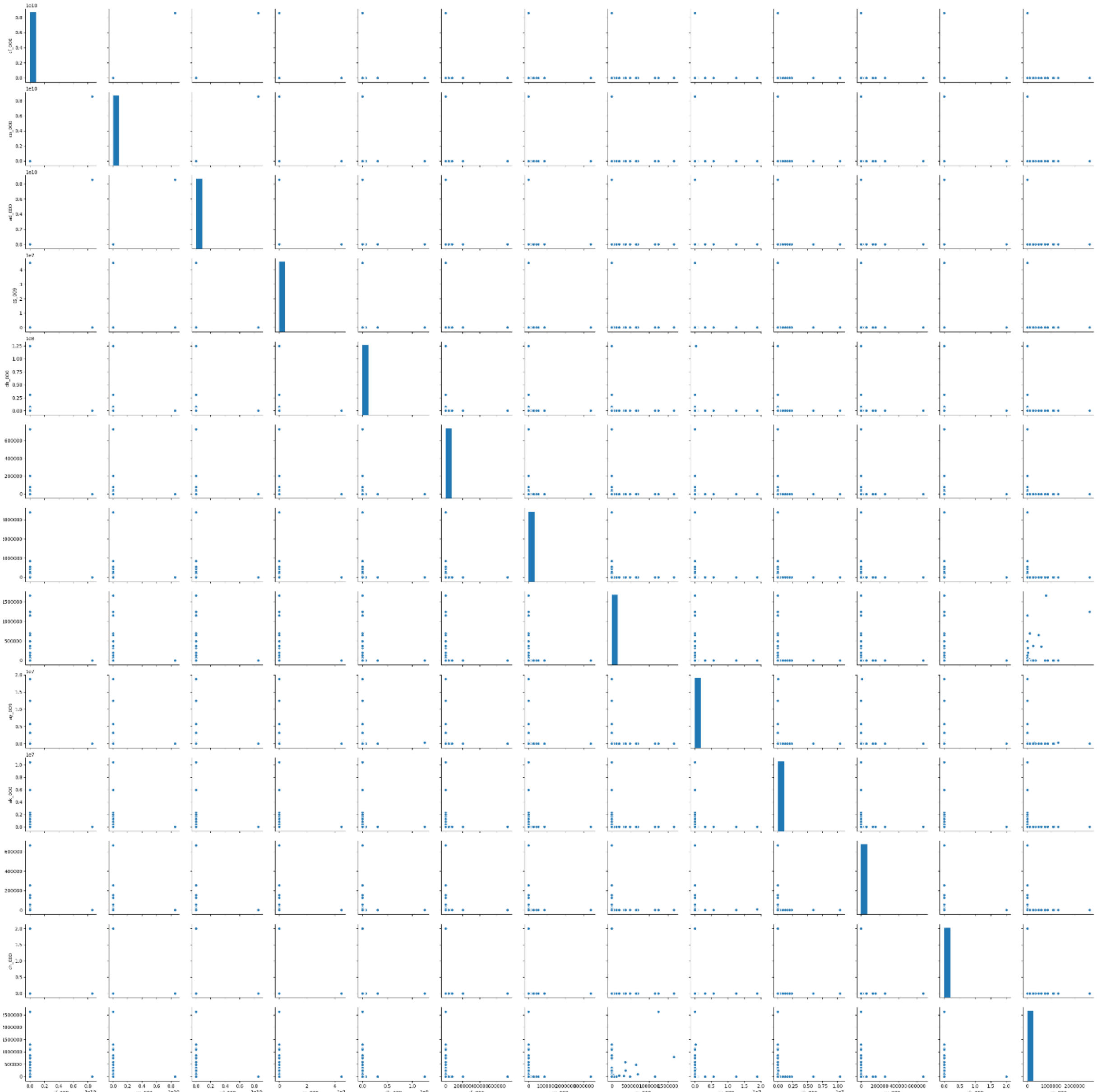
(iv) Box-plots and Scatter Plot

Top 13 features:
['cf_000', 'co_000', 'ad_000', 'cs_009', 'dh_000', 'dj_000', 'ag_000', 'as_000', 'ay_009', 'ak_000', 'az_009', 'ch_000', 'au_000']





It is difficult to draw any conclusions just yet with only the scatter-matrix.
We might need to study the coefficients of the parameters in the model to further explore their importance.



(v) Imbalance in data

Yes, the dataset is heavily imbalanced as the _neg_ samples outweigh the _pos_ ones.

This is clearly evident from the following results.

Training set:

neg 59000

Name: classs, dtype: int64

pos 1000

Name: classs, dtype: int64

Testing set:

neg   15625

Name: classs, dtype: int64

pos   375

Name: classs, dtype: int64

(c)

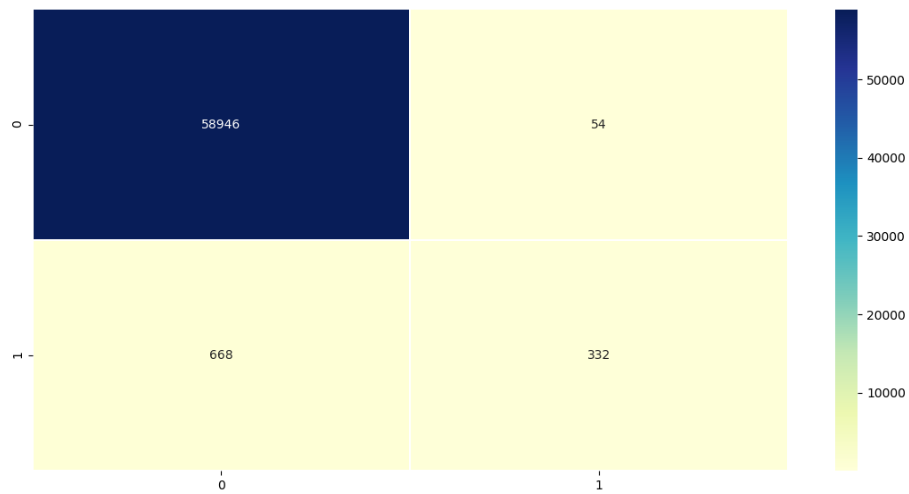
Random Forest Classifier with Imbalance

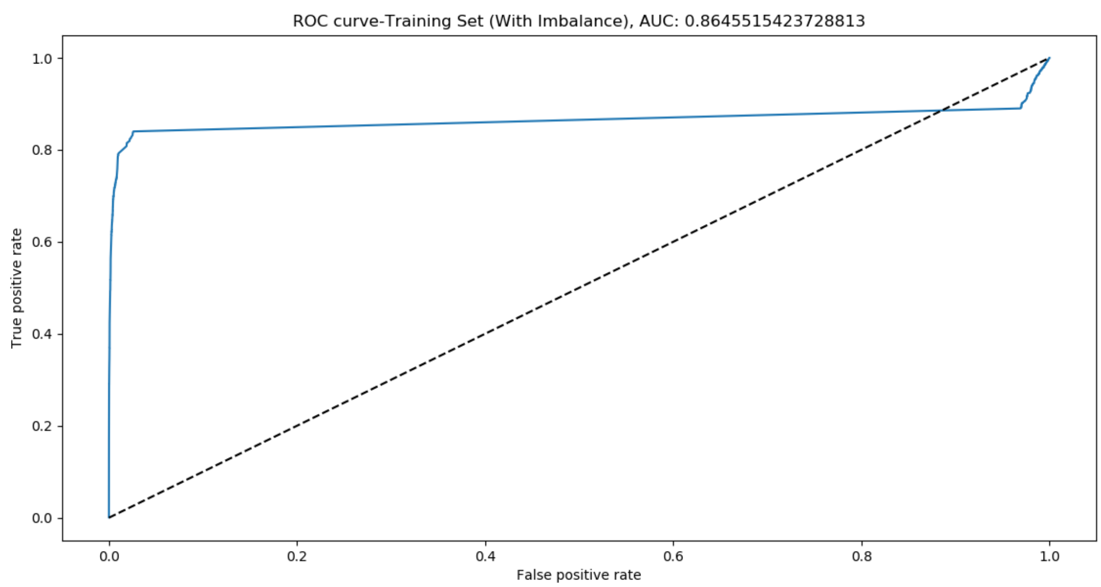
Train Error:

missclassification rate: 0.012033333333333333

oob_error 0.012900000000000023

The oob_error is slightly better than the regular misclassification error.



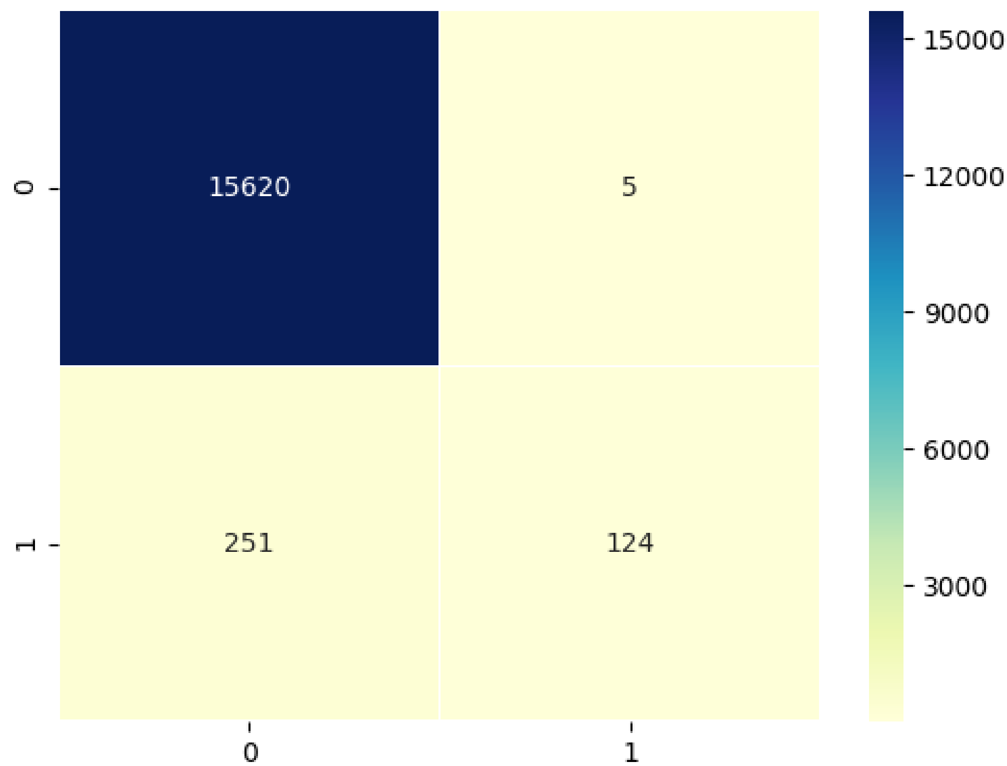


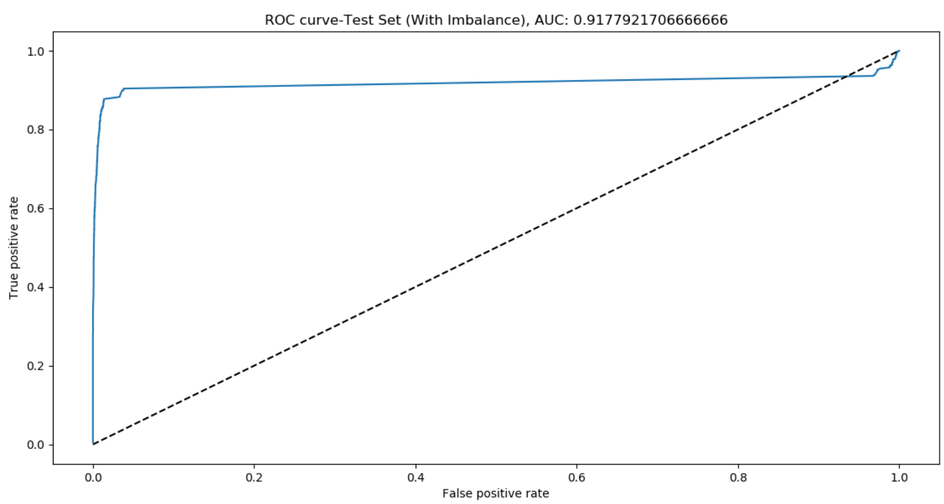
Test Error:

missclassification rate: 0.016

oob_error 0.012549999999999995

Here again the oob_error is better as compared to test error.


15620
5
251
124
0
1
0
1
15000
12000
9000
6000
3000


ROC curve-Test Set (With Imbalance), AUC: 0.9177921706666666
True positive rate
False positive rate
0.0
0.2
0.4
0.6
0.8
1.0

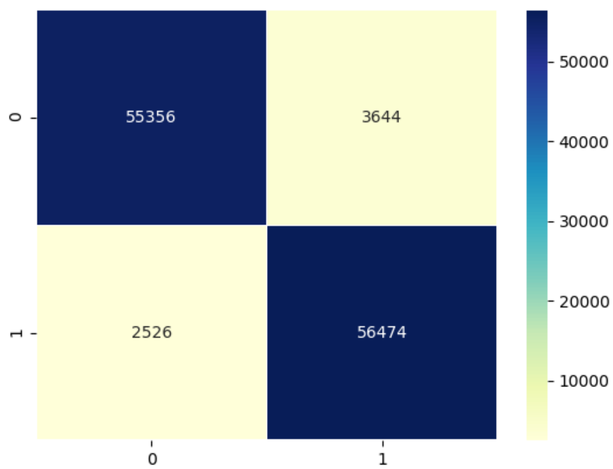
(d) RF with Imbalance Removal

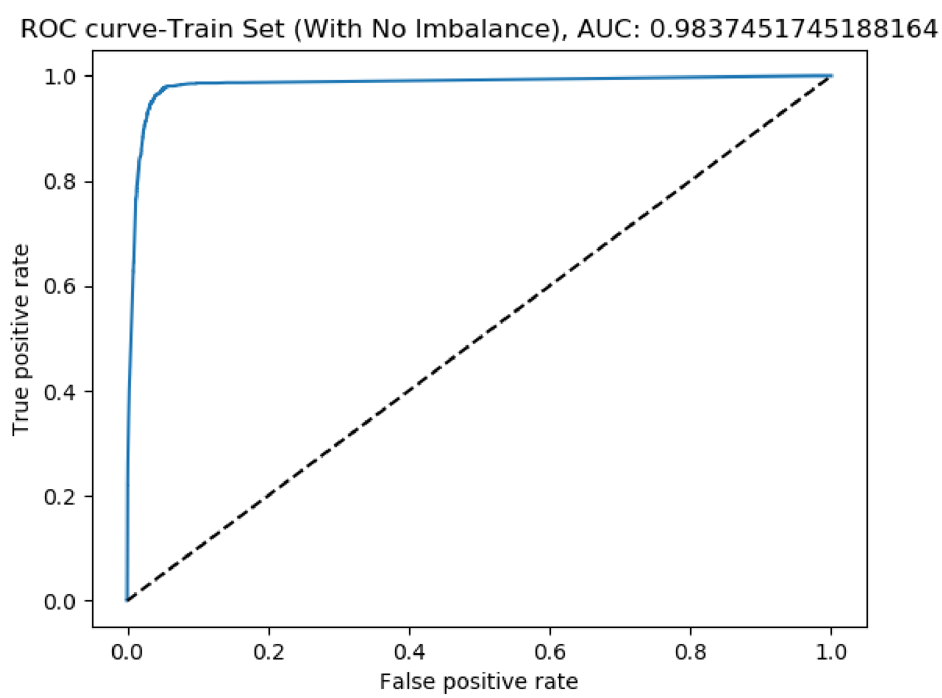
Resampling, downsampling and upsampling are some of the ways to deal with imbalance.

Train Set error:

missclassification rate: 0.05228813559322034

OOB error: 0.0566186440677966

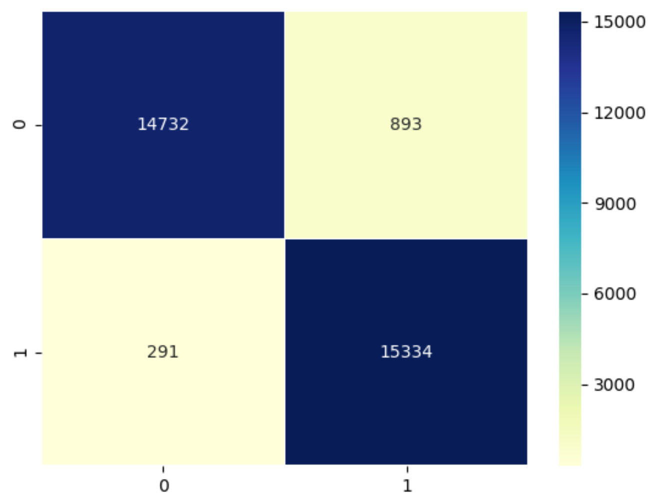


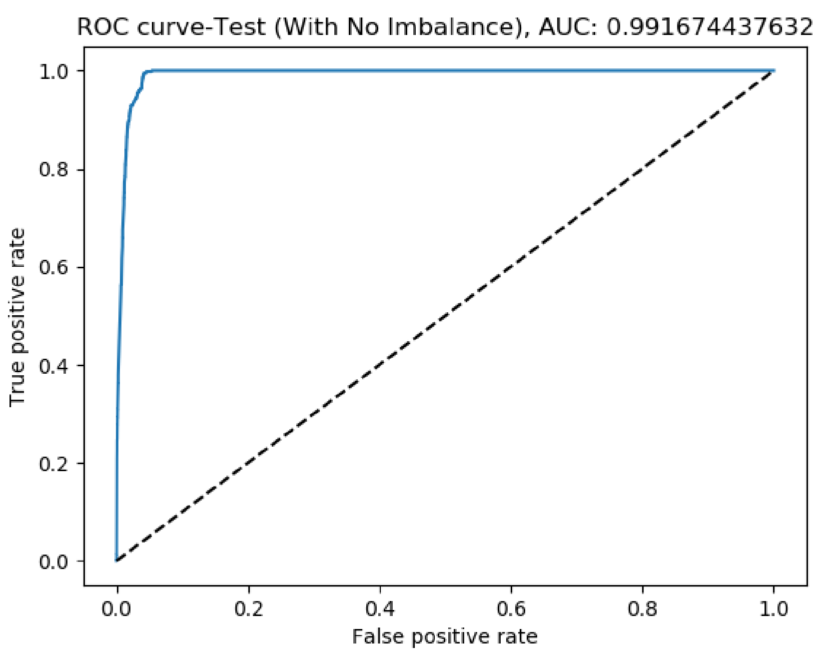


Test Set:

missclassification rate: 0.037888

OOB Error: 0.05411016949152547





Here the Misclassification rate outperforms the OOB_error

Clearly the resampling helps the model and the error is reduced significantly as compared to 2c.

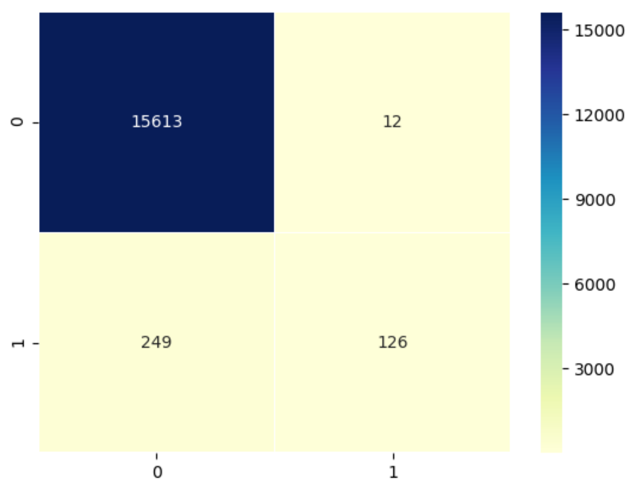
(e) Model Trees with weka classifier (LMT)

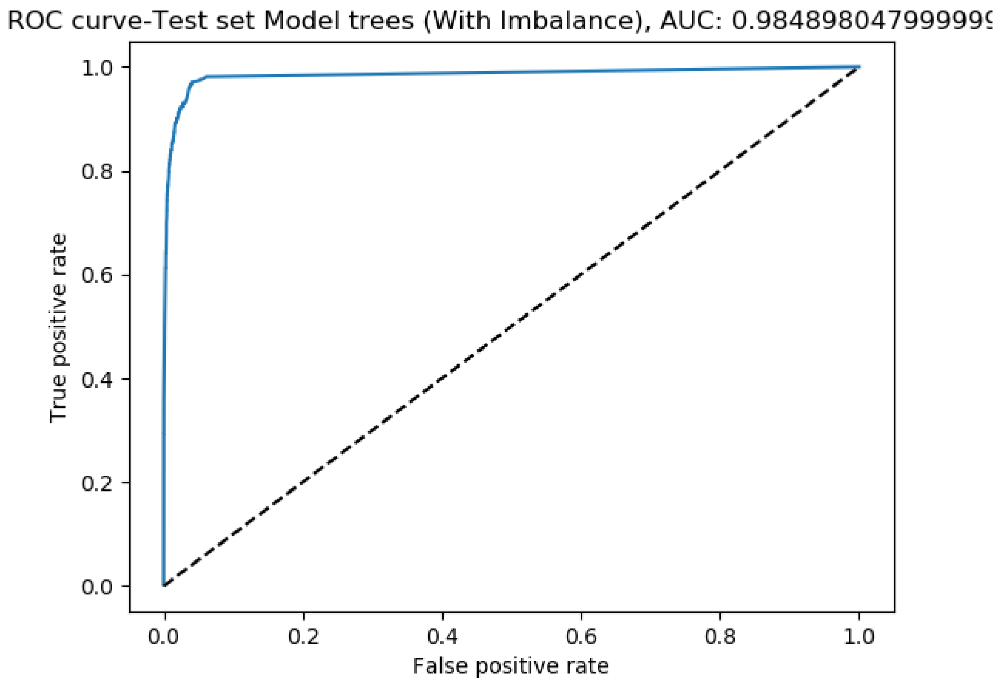
I used the weka.classifier and used LMT in that as the value for classes.

weka.classfier.trees.LMT

Test Set:

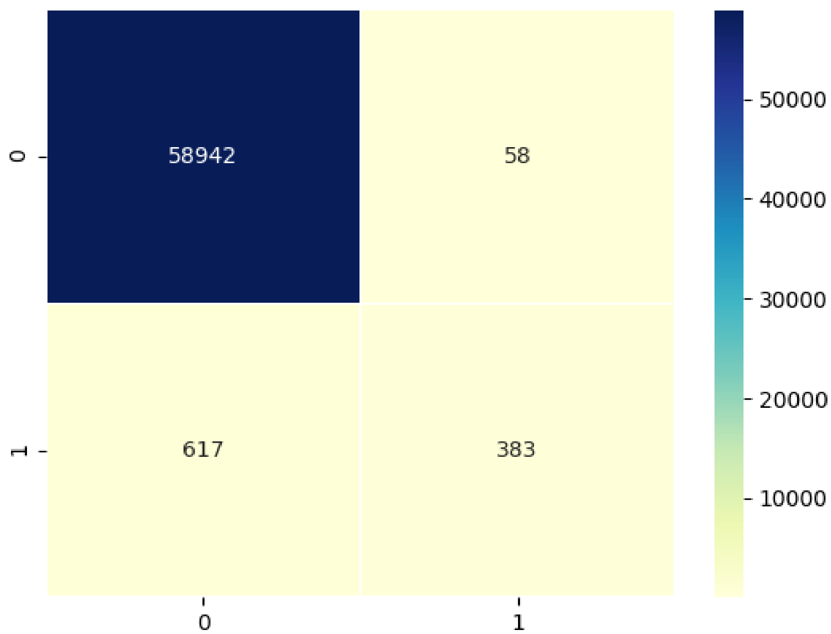
misclassification rate: 0.0163125

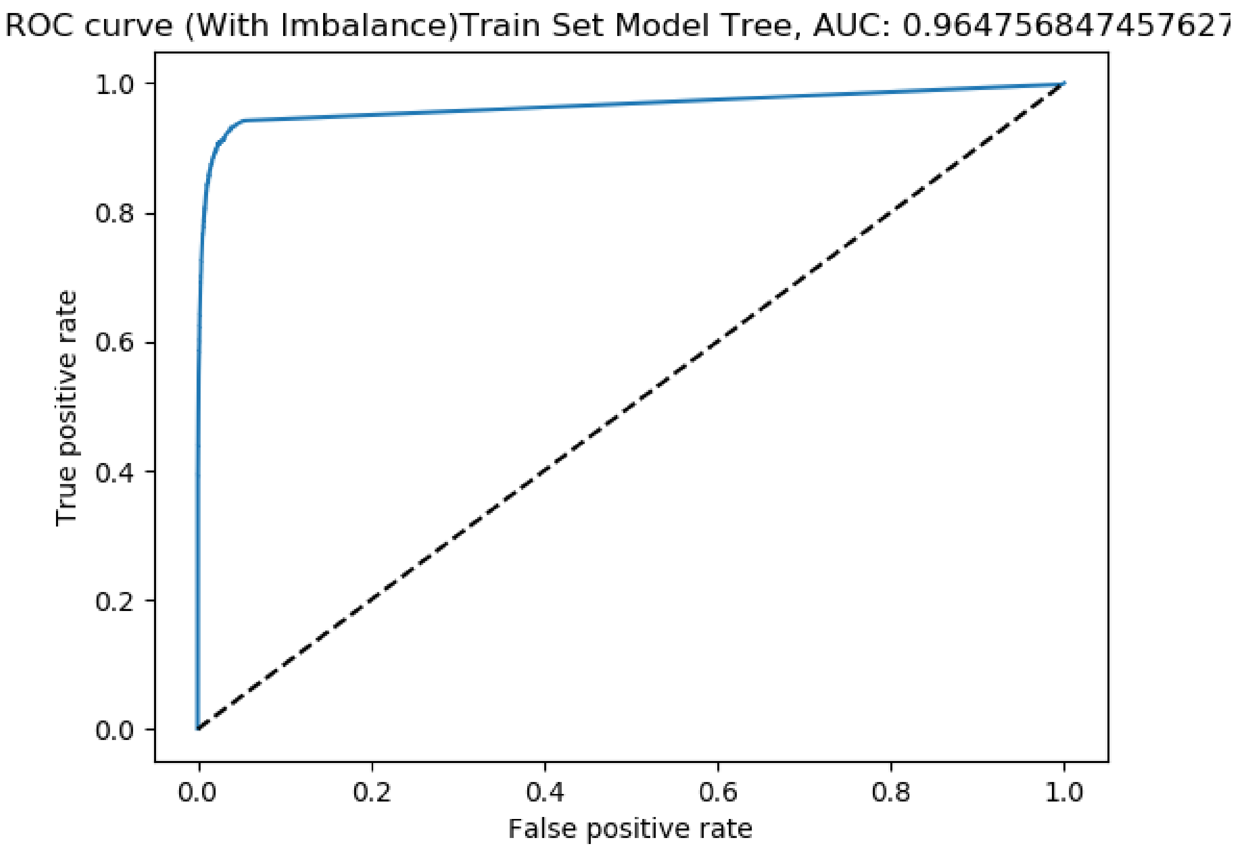




Train set:

misclassification rate: 0.01125


58942
58
617
383
0
1
0
1
50000
40000
30000
20000
10000

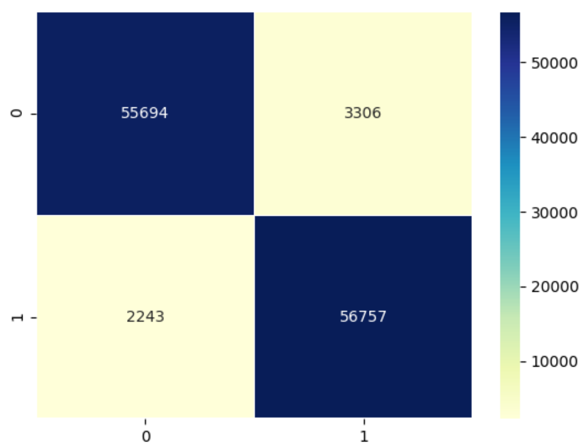

ROC curve (With Imbalance)Train Set Model Tree, AUC: 0.964756847457627
True positive rate
False positive rate
0.0
0.2
0.4
0.6
0.8
1.0

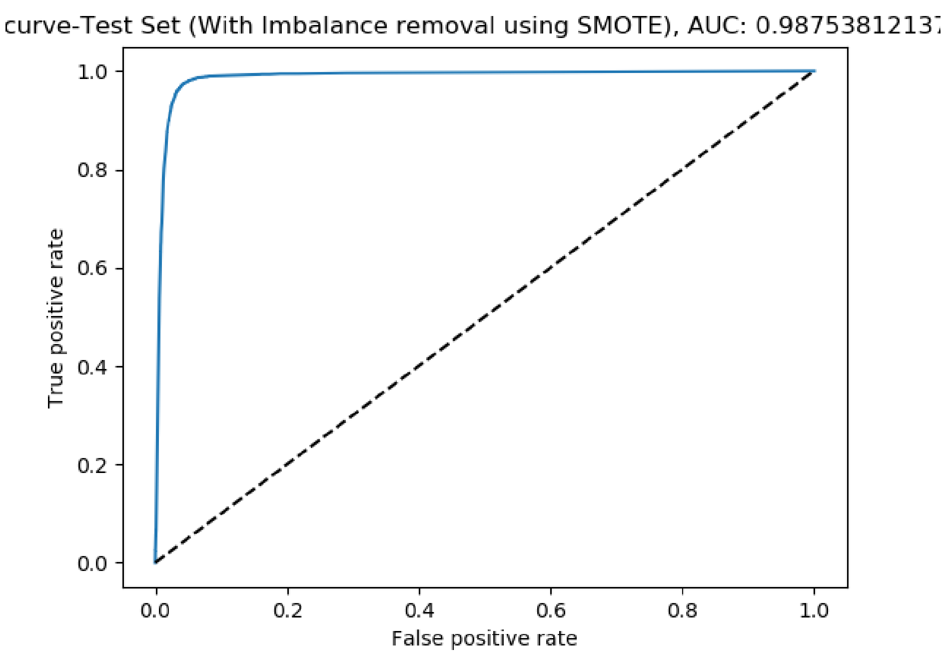
(f) Model trees with SMOTE Filter

I then used a Filter from the weka wrapper balance the dataset.

- **I used** weka.filters.supervised.instance.SMOTE and plugged this into the Filter.

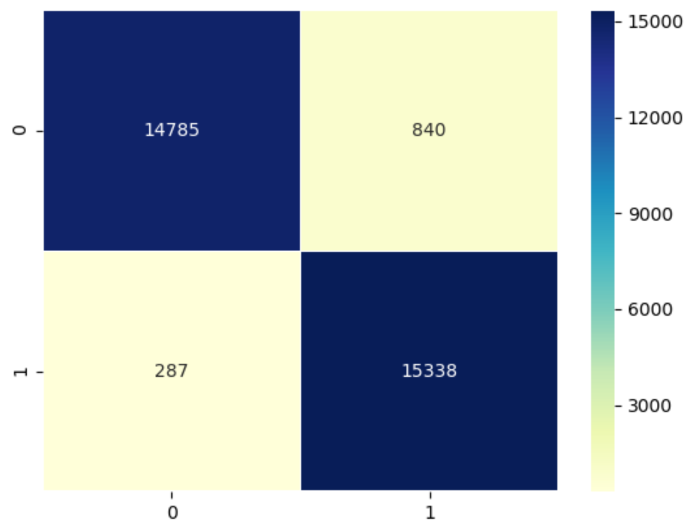
misclassification rate: 0.04755084745762712

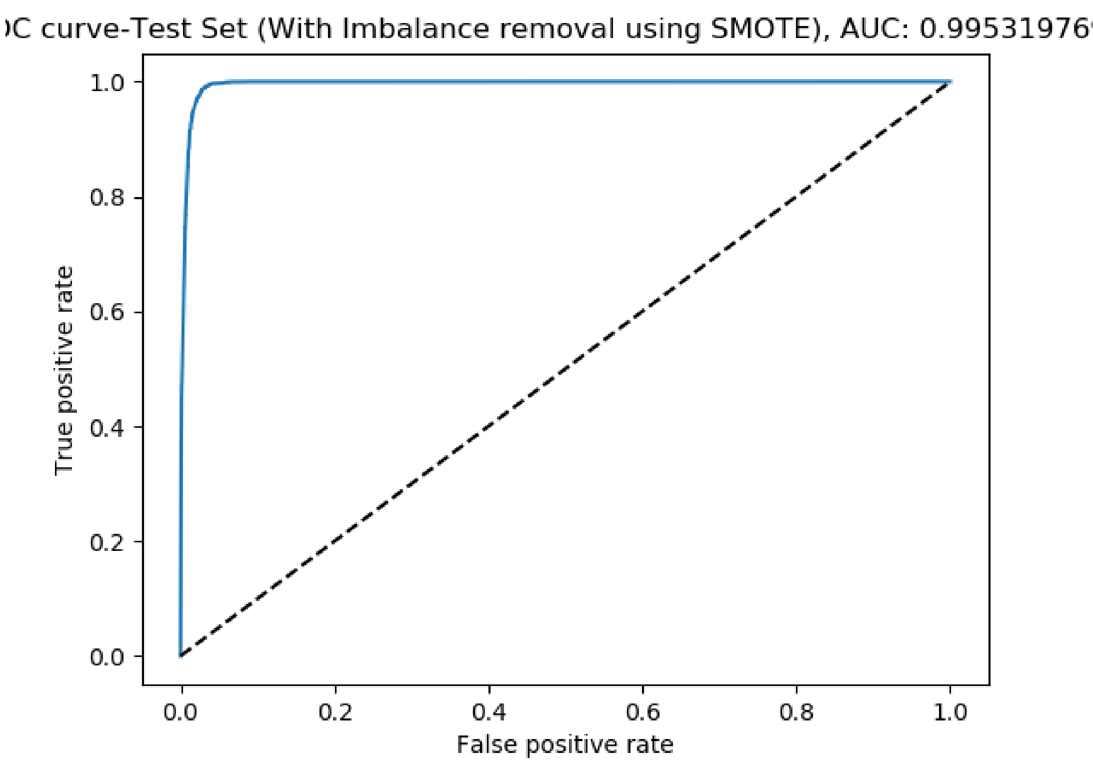




Test Set:

misclassification rate: 0.036064





The Smote technique to sample has definitely worked its magic as the error rate is cut down significantly and the AUC is increased, which means a better model.

ISLR Questions

USC ID: 5423417955

Question - 3 - ISLR- 6.8.3.

(a) It will decrease steadily. As with increase s, the constraint on β_j is less strict thus model becomes flexible and RSS (training) decreases.

(b) Decrease initially, and then start increasing in U-shape. As we increase s, β_j has lesser restriction, thus again the flexibility of model increases thus RSS decreases to start with but then increases in a U shape.

(c) Steadily increases. As s increases model becomes more flexible thus results in steady increase in variance.

(d) Steadily decreases. As we increase s with 0. Thus, the model becomes more flexible this leads to decrease in bias.

(e) Remains constant. As this irreducible error is independant of the model.

Question-4 : ISLR 6.8.5.

(a) Given,

$x_{11} = x_{12} = x_1$,

$x_{21} = x_{22} = x_2$.

In Ridge regression, we try to minimize:

$$(y_1 - \hat{\beta}_1 x_1 - \hat{\beta}_2 - x_1)^2 + (y_2 - \hat{\beta}_1 x_2 - \hat{\beta}_2 x_2) + \lambda(\hat{\beta}_1^2 + \hat{\beta}_2^2)$$

(b) Differentiating wrt $\hat{\beta}_1, \hat{\beta}_2$ and equating to 0. we get,

$$\hat{\beta}_1 (x_1^2 + x_2^2 + \lambda) + \hat{\beta}_2 (x_1^2 + x_2^2) = y_1 x_1 + y_2 x_2 \quad (1)$$

And,

$$\hat{\beta}_1 (x_1^2 + x_2^2) + \hat{\beta}_2 (x_1^2 + x_2^2 + \lambda) = y_1 x_1 + y_2 x_2 \quad (2)$$

(1) - (2)

we obtain

$$\hat{\beta}_1 = \hat{\beta}_2.$$

(c) According to Lasso,

$x_{11} = x_{12} = x_1$;

$x_{21} = x_{22} = x_2$.

We would like to minimize :

$$\left((y_1 - \hat{\beta}_1 x_1) - \hat{\beta}_2 x_1\right)^2 + (y_2 - \hat{\beta}_1 x_2 - \hat{\beta}_2 x_2)^2 + \lambda\left(|\hat{\beta}_1| + |\hat{\beta}_2|\right).$$

(d) We can say that,

$$(y_1 - \hat{\beta}_1 x_1 - \hat{\beta}_2 x_1)^2 + (y_2 - \hat{\beta}_1 x_2 - \hat{\beta}_2 x_2)^2$$

given, $\sum_{i=1}^{n} |\hat{\beta}_i| \leq s$.

The lasso constraint takes the shape of a diamond with center at origin of $(\hat{\beta}_1, \hat{\beta}_2)$.

Thus, if $x_{11} = x_{12} = x_1$,

$x_{21} = x_{22} = x_2$,

$x_1 + x_2 = 0$, $y_1 + y_2 = 0$.

We look to minimize ;

$$2\left[y_1 - (\hat{\beta}_1 + \hat{\beta}_2 x_1)\right]^2 \geq 0 .$$

A unique solution $\hat{\beta}_1 + \hat{\beta}_2 = \frac{y_1}{x_1}$ exists.

This is parallel to the edge of diamond of constraints of $[y_1 - (\hat{\beta}_1 + \hat{\beta}_2) x_1]^2$ intersects the diamond of constraints. So, the edge $\hat{\beta}_1 + \hat{\beta}_2 = s$ is also a solution.

Thus the optimization problem has many possible solutions.

Question 5 : ISLR 8.4.5

If we use majority voting; we will be classifying X as Red as it has 6 red & 4 green, thus making red more commonly occurring.

However, if we use average probability method we classify X into Green, because the average of 10 probs. is 0.45.

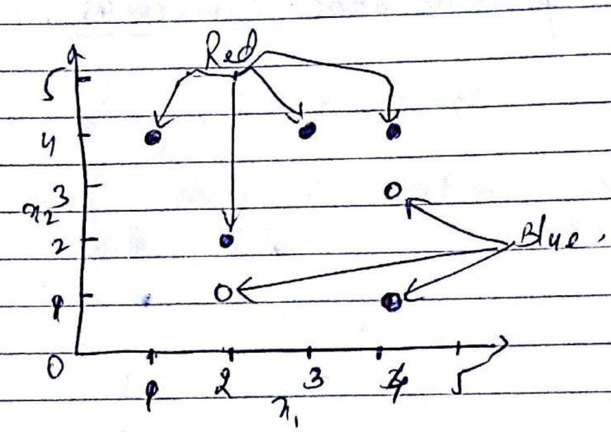
Question 6 : ISLR 9.7.3.

(a) $x_1 = c(3, 2, 4, 1, 2, 4, 4)$

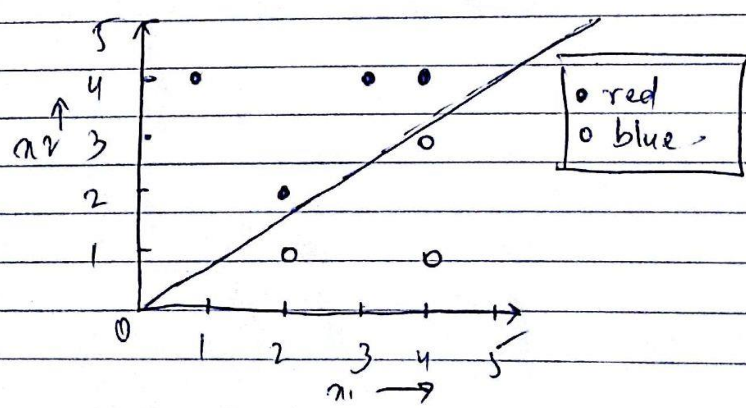
$x_2 = c(4, 2, 4, 4, 1, 3, 1)$

colors = c("red", "red", "red", "red", "blue", "blue", "blue")

plot (x_1, x_2, col = colors, xlim = c(0,5), ylim = c(0,5)).



(b)



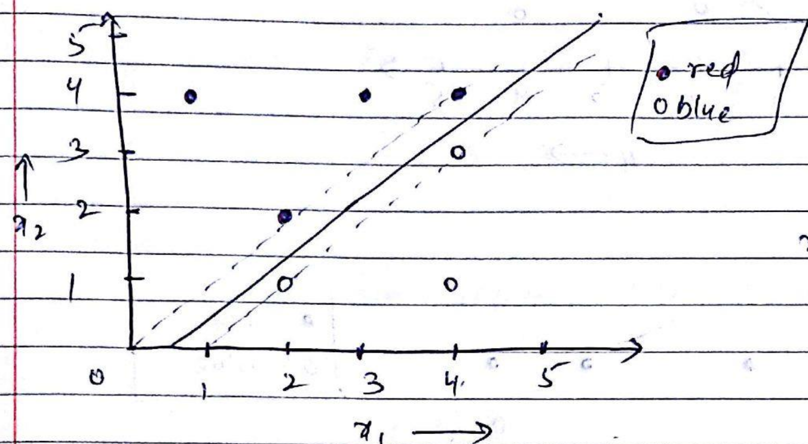
(c) If $x_1 - x_2 - 0.5 < 0$

Classify as Red.

Else

Classify as Blue

(d)

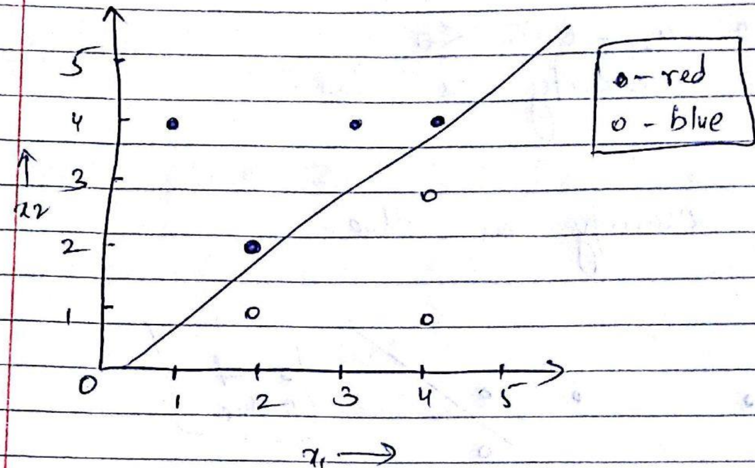


margin = 1/4

(e) Support vectors : (2, 1), (2, 2), (4, 4), (4, 3).

(f) If we moved (4, 1), we would not change the maximal hyperplane and it is not a SV.

(g) $x_1 - x_2 - 0.3 = 0$ is not an optimal separating hyperplane.



(h)